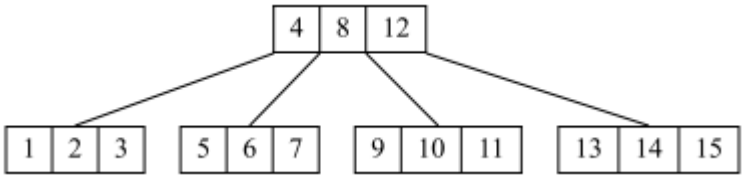
Solution 1 **(2-4 Trees –** ***4 points*****)**

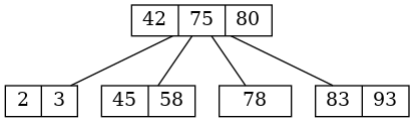
1. The smallest 2-3-4 tree containing the integers in $[1, 15]$:



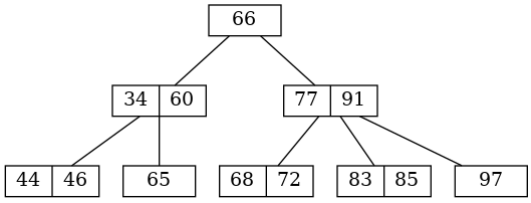
2. Minimal height of a 2-4 tree containing 63 keys: 2 (only 4-nodes)

3. Maximum height of a 2-4 tree containing 63 keys: 5 (only 2-nodes = binary tree)

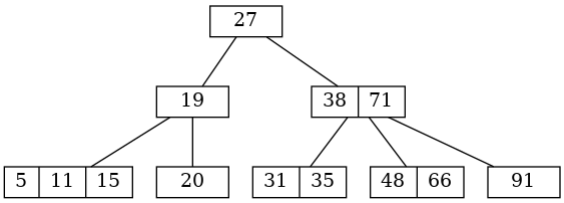
4. 24 trees:



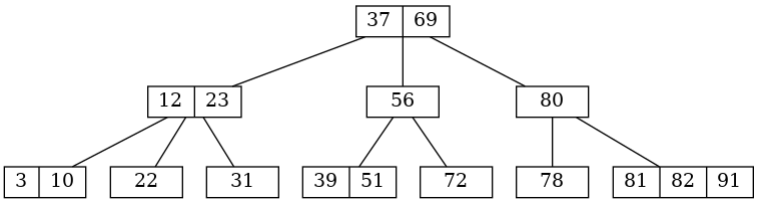
OUI



NON



OUI

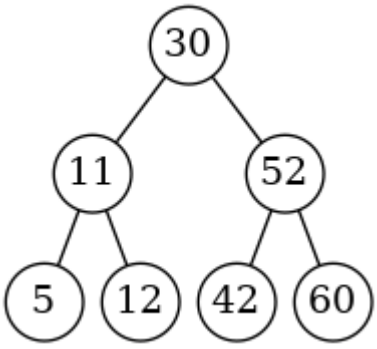


NON

Solution **(Drawings – *4 points*)**

1. *Insertions*

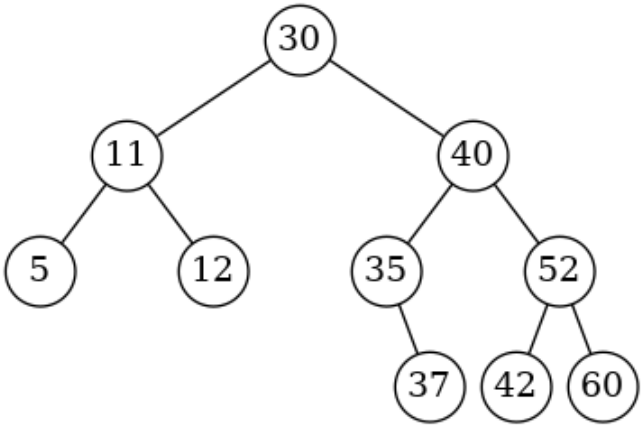
Tree built by insertions of 42, 30, 11, 5, 12, 60, 52:



Rotations:

rr(42) / rd(42)
rlr(42) / rdg(42)

Tree after insertions of 35, 40, 37 :

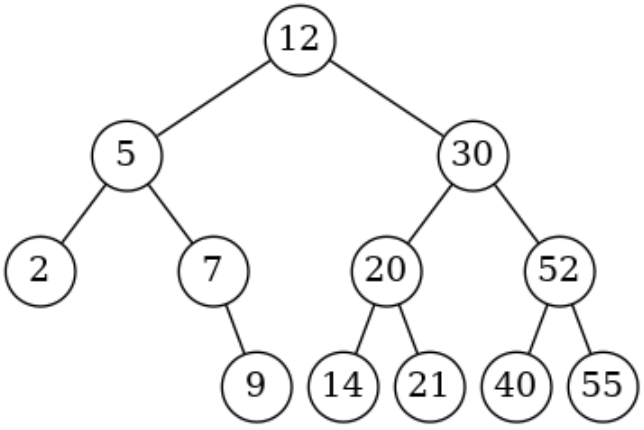


Rotations:

lrr(42) / rgd(42)
rr(52) / rd(52)

2. *Deletion*

Tree after deletion of 50:



Rotations:

rlr(40) / rdg(40) (ok si 50)
rr(30) / rd(30)

***Solution 3* (Depth insertion– *6 points*)**

The function `insert_prof(`B`, `x`)` inserts in leaf the key `x` in the binary search tree `B` unless it is already in the tree and returns a pair `(root, d)` where:

- `root` is the resulting binary search tree
- `d` is the depth where `x` has been inserted or -1 if `x` is already in the tree

```
    def aux_prof(B, x, p):
        if B == None:
            return BinTree(x, None, None), p
        else:
            if x == B.key:
                return B, -1
            else:
                if x < B.key:
                    B.left, res = aux_prof(B.left, x, p+1)
                else:
                    B.right, res = aux_prof(B.right, x, p+1)
                return B, res

    def insert_prof(B, x):
        return aux_prof(B, x, 0)
```

```
    def insert_prof(B, x):
        if B == None:
            return BinTree(x, None, None), 0
        else:
            if x == B.key:
                return B, -1
            else:
                if x < B.key:
                    B.left, res = insert_prof(B.left, x)
                else:
                    B.right, res = insert_prof(B.right, x)
                if res == -1:
                    return B, res
                else:
                    return B, res+1
```

```
    def aux_prof(B, x, d):
        if x == B.key:
            return -1
        else:
            if x < B.key:
                if B.left == None:
                    B.left = BinTree(x, None, None)
                    return d+1
                else:
                    return aux_prof(B.left, x, d+1)
            else:
                if B.right == None:
                    B.right = BinTree(x, None, None)
                    return d+1
                else:
                    return aux_prof(B.right, x, d+1)

    def insert_prof(B, x):
        if B == None:
            return BinTree(x, None, None), 0
        else:
            return B, aux_prof(B, x, 0)
```

Solution 4 **(Second minimum – *6 points*)**

WThe function `second_min(`B`)` returns the second smallest value (2nd in increasing order) of the binary search tree `B` or the value `None` if it does not exist. All the keys of the binary search tree `B` are assumed distinct.

```
def second_min(B):
    if B == None:
        return None
    else:
        anc = None
        while B.left != None:
            anc = B
            B = B.left
        if B.right == None:
            if anc == None:
                return None
            else:
                return anc.key
        else:
            B = B.right
            while B.left != None:
                B = B.left
            return B.key
```

```
def aux_min(B):
    if B.left == None:
        return B.key
    else:
        return aux_min(B.left)

def aux_second(B, anc):
    if B.left == None:
        if B.right == None:
            return anc.key
        else:
            return aux_min(B.right)
    else:
        return aux_second(B.left, B)

def second_min(B):
    if B == None:
        return None
    else:
        return aux_second(B, None)
```

```
def __min(B):
    while B.left != None:
        B = B.left
    return B

def __second_min(B):
    if B.left == None:
        if B.right != None:
            return __min(B.right).key
        else:
            return None
    else:
        m = __second_min(B.left)
        if m == None:
            return B.key
        else:
            return m

def second_min(B):
    if B == None:
        return None
    else:
        return __second_min(B)
```